Extending Hive

Chapter 12.3

Extending Hive

- What role does SerDes play in Hive?
- How to use a custom SerDe
- How to use `TRANSFORM` for custom record processing
- How to add support for a User-Defined Function (UDF)
- How to use variable substitution

© Copyright 2010-2014 Cloudera. All rights reserved. Not to be reproduced without prior written consent.

Chapter Topics

Extending Hive

- **SerDes**
- Data Transformation with Custom Scripts
- User-Defined Functions
- Parameterized Queries

© Copyright 2010-2014 Cloudera. All rights reserved. Not to be reproduced without prior written consent.

Hive SerDes

- **Hive uses a SerDe for reading and writing records**
  - Stands for "Serializer / Deserializer"
  - SerDes control the row format of the table
  - Specified, sometimes implicitly, when table is created
- **Hive ships with many SerDes, including:**

Name	Reads and Writes Records
`LazySimpleSerDe`	Using specified field delimiters (default)
`RegexSerDe`	Based on supplied patterns
`ColumnarSerDe`	Using the columnar format needed by RCFile
`HBaseSerDe`	Using an HBase table

© Copyright 2010-2014 Cloudera. All rights reserved. Not to be reproduced without prior written consent.

Recap: Creating a Table with Regex SerDe

Input Data

```
05/23/2013 19:45:19 312-555-7834 CALL_RECEIVED ""
05/23/2013 19:48:37 312-555-7834 COMPLAINT "Item not received"
```

Using SerDe

```
CREATE TABLE calls (
   event_date STRING,
   event_time STRING,
   event_type STRING,
   phone_num STRING,
   details STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.RegexSerDe'
WITH SERDEPROPERTIES ("input.regex" =
      "([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) \"([^\"]*)\"");
```

Resulting Table

event_date	event_time	event_type	phone_num	details
05/23/2013	19:45:19	312-555-7834	CALL_RECEIVED	
05/23/2013	19:45:37	312-555-7834	COMPLAINT	Item not received

© Copyright 2010-2014 Cloudera. All rights reserved. Not to be reproduced without prior written consent.

Adding a Custom SerDe to Hive

- **Hive also allows writing custom SerDes using its Java API**
  - There are many open source SerDes on the Web
  - Writing your own is seldom necessary
- **We will now explain how to add a custom SerDe to Hive**
  - It reads and writes records in CSV format
  - Using JAR file from `http://tiny.cloudera.com/dac14a`

© Copyright 2010-2014 Cloudera. All rights reserved. Not to be reproduced without prior written consent.

Adding a JAR File to Hive

- **You must register external libraries before using them**
  - Ensures Hive can find the library (JAR file) at runtime

```
hive> ADD JAR /home/training/csv-serde-1.0.jar;
```

- **Remains in effect only during the current Hive session**
  - Consider editing your `.hiverc` file to add frequently-used JARs

© Copyright 2010-2014 Cloudera. All rights reserved. Not to be reproduced without prior written consent.

Using the SerDe in Hive

Input Data

```
1,Gigabux,gigabux@example.com
2,"ACME Distribution Co.",acme@example.com
3,"Bitmonkey, Inc.",bmi@example.com
```

Specify SerDe

```
CREATE TABLE vendors
    (id INT,
     name STRING,
     email STRING)
  ROW FORMAT SERDE 'com.bizo.hive.serde.csv.CSVSerde';
```

Resulting Table

id	name	email
1	Gigabux	gigabux@example.com
2	ACME Distribution Co.	acme@example.com
3	Bitmonkey, Inc.	bmi@example.com

© Copyright 2010-2014 Cloudera. All rights reserved. Not to be reproduced without prior written consent.

Chapter Topics

Extending Hive

- SerDes
- **Data Transformation with Custom Scripts**
- User-Defined Functions
- Parameterized Queries

© Copyright 2010-2014 Cloudera. All rights reserved. Not to be reproduced without prior written consent.

Using TRANSFORM to Process Data Using External Scripts

- **You are not limited to manipulating data exclusively in HiveQL**
  - Hive allows you to transform data through external scripts or programs
  - These can be written in nearly any language
- **This is done with HiveQL's `TRANSFORM . . . USING` construct**
  - One or more fields are supplied as arguments to `TRANSFORM()`
  - The external script is identified by `USING`
    - It receives each record, processes it, and returns the result
- **Use `ADD FILE` to distribute the script to nodes in the cluster**

```
hive> ADD FILE myscript.pl;
hive> SELECT TRANSFORM(*) USING 'myscript.pl'
      FROM employees;
```


© Copyright 2010-2014 Cloudera. All rights reserved. Not to be reproduced without prior written consent.

Data Input and Output with TRANSFORM

- **Your external program will receive one record per line on standard input**
  - Each field in the supplied record will be a tab-separated string
  - NULL values are converted to the literal string \N
- **You may need to convert values to appropriate types within your program**
  - For example, converting to numeric types for calculations
- **Your program must return tab-delimited fields on standard output**
  - Output fields can optionally be named and cast using the syntax below

```
SELECT TRANSFORM(product_name, price)
       USING 'tax_calculator.py'
       AS (item_name STRING, tax INT)
       FROM products;
```


© Copyright 2010-2014 Cloudera. All rights reserved. Not to be reproduced without prior written consent.

Hive TRANSFORM Example (1)

- **Here is a complete example of using TRANSFORM in Hive**
  - Our Perl script parses an e-mail address, determines to which country it corresponds, and then returns an appropriate greeting
  - Here's a sample of the input data

```
hive> SELECT name, email_address FROM employees;
Antoine  antoine@example.fr
Kai      kai@example.de
Pedro    pedro@example.mx
```

  - Here's the corresponding HiveQL code

```
hive> ADD FILE greeting.pl;
hive> SELECT TRANSFORM(name, email_address)
        USING 'greeting.pl' AS greeting
        FROM employees;
```


© Copyright 2010-2014 Cloudera. All rights reserved. Not to be reproduced without prior written consent.

Hive TRANSFORM Example (2)

- **The Perl script for this example is shown below**
  - A complete explanation of this script follows on the next few slides

```
#!/usr/bin/env perl

%greetings = ('de' => 'Hallo',
              'fr' => 'Bonjour',
              'mx' => 'Hola');

while (<STDIN>) {
   ($name, $email) = split /\t/;
   ($suffix) = $email =~ /\.([a-z]+)/;
   $greeting = $greetings{$suffix};
   $greeting = 'Hello' unless defined($greeting);
   print "$greeting $name\n";
}
```


© Copyright 2010-2014 Cloudera. All rights reserved. Not to be reproduced without prior written consent.

Hive TRANSFORM Example (3)

```
#!/usr/bin/env perl

%greetings = ('de' => 'Hallo',
              'fr' => 'Bonjour',
              'mx' => 'Hola');

while (<
    ($nam
    ($suf
    $gree
    $gree
    print
}
```

The first line tells the system to use the Perl interpreter when running this script.

We define our greetings in the next line using an associative array keyed by the country code we'll extract from the e-mail address.

© Copyright 2010-2014 Cloudera. All rights reserved. Not to be reproduced without prior written consent.

Hive TRANSFORM Example (4)

```
#!/usr/bin/env perl

%greeti

while (<STDIN>) {
    ($name, $email) = split /\t/;
    ($suffix) = $email =~ /\.([a-z]+)/;
    $greeting = $greetings{$suffix};
    $greeting = 'Hello' unless defined($greeting);
    print "$greeting $name\n";
}
```

We read each record from standard input within the loop, and then split them into fields based on tab characters.

© Copyright 2010-2014 Cloudera. All rights reserved. Not to be reproduced without prior written consent.

Hive TRANSFORM Example (5)

```
#!/usr/bin/env perl

%greetings = ('de' => 'Hallo',
              'fr' => 'Bonjour',
              'mx' => 'Hola');

while (<STDIN>) {
   ($name, $email) = split /\t/;
   ($suffix) = $email =~ /\.([a-z]+)$/;
   $greeting = $greetings{$suffix};
   $greeting = 'Hello' unless defined($greeting);

}
```

We extract the country code from the e-mail address (the pattern matches any letters following the final dot). We use that to look up a greeting, but default to 'Hello' if we didn't find one.

© Copyright 2010-2014 Cloudera. All rights reserved. Not to be reproduced without prior written consent.

Hive TRANSFORM Example (6)

```
#!/usr/bin/env perl

%greetings = ('de' => 'Hallo',
              'fr' => 'Bonjour',

wh
    $greeting = 'Hello' unless defined($greeting);
    print "$greeting $name\n";
}
```

Finally, we return our greeting as a single field by printing this value to standard output. If we had multiple fields, we'd simply separate each by tab characters when printing them here.

© Copyright 2010-2014 Cloudera. All rights reserved. Not to be reproduced without prior written consent.

Hive TRANSFORM Example (7)

- **Finally, here's the result of our transformation**

```
hive> ADD FILE greeting.pl;
hive> SELECT TRANSFORM(name, email_address)
          USING 'greeting.pl' AS greeting
          FROM employees;

Bonjour Antoine
Hallo Kai
Hola Pedro
```


© Copyright 2010-2014 Cloudera. All rights reserved. Not to be reproduced without prior written consent.

Chapter Topics

Extending Hive

- SerDes
- Data Transformation with Custom Scripts
- **User-Defined Functions**
- Parameterized Queries

© Copyright 2010-2014 Cloudera. All rights reserved. Not to be reproduced without prior written consent.

Overview of User-Defined Functions (UDFs)

- **User-Defined Functions (UDFs) are custom functions**
  - Invoked with the same syntax as built-in functions

```
hive> SELECT CALC_SHIPPING_COST(order_id, 'OVERNIGHT')
      FROM orders WHERE order_id = 5742354;
```

- **There are three types of UDFs in Hive**
  - Standard UDFs
  - User-Defined Aggregate Functions (UDAFs)
  - User-Defined Table Functions (UDTFs)

© Copyright 2010-2014 Cloudera. All rights reserved. Not to be reproduced without prior written consent.

Developing Hive UDFs

- **Hive UDFs are written in Java**
  - Currently no support for writing UDFs in other languages
  - Using `TRANSFORM` may be an alternative to UDFs
- **Open source UDFs are plentiful on the Web**
- **There are three steps for using a UDF in Hive**
  1. Add the function's JAR file to Hive
  2. Register the function itself
  3. Use the function in your query

Attention Java Developers
Cloudera now offers a free e-learning module "Writing UDFs for Hive"

`http://tiny.cloudera.com/dac14b`

© Copyright 2010-2014 Cloudera. All rights reserved. Not to be reproduced without prior written consent.

Example: Using a UDF in Hive (1)

- **Our example UDF was compiled from sources found in GitHub**
  - Popular Web site for many open source software projects
  - Project URL: `http://tiny.cloudera.com/dac14e`
- **We compiled the source and packaged it into a JAR file**
  - We have included a copy of it on your VM
- **Our example shows the `DATE_FORMAT` UDF in that JAR file**
  - Allows great flexibility in formatting date fields in output

© Copyright 2010-2014 Cloudera. All rights reserved. Not to be reproduced without prior written consent.

Example: Using a UDF in Hive (2)

- **First, register the JAR with Hive**
  - Same step as with a custom SerDe

```
hive> ADD JAR date-format-udf.jar;
```

- **Next, register the function and assign an alias**
  - The quoted value is the fully-qualified Java class for the UDF

```
hive> CREATE TEMPORARY FUNCTION DATE_FORMAT
      AS 'com.nexr.platform.hive.udf.UDFDateFormat';
```


© Copyright 2010-2014 Cloudera. All rights reserved. Not to be reproduced without prior written consent.

Example: Using a UDF in Hive (3)

- **You may then use the function in your query**

```
hive> SELECT order_date FROM orders LIMIT 1;
2011-12-06 10:03:35

hive> SELECT DATE_FORMAT(order_date, 'dd-MMM-yyyy')
      FROM orders LIMIT 1;
06-Dec-2011

hive> SELECT DATE_FORMAT(order_date, 'dd/mm/yy')
      FROM orders LIMIT 1;
06/12/11

hive> SELECT DATE_FORMAT(order_date, 'EEEE, MMM d, yyyy')
      FROM orders LIMIT 1;
Tuesday, Dec 6, 2011
```


© Copyright 2010-2014 Cloudera. All rights reserved. Not to be reproduced without prior written consent.

Chapter Topics

Extending Hive

- SerDes
- Data Transformation with Custom Scripts
- User-Defined Functions
- **Parameterized Queries**

© Copyright 2010-2014 Cloudera. All rights reserved. Not to be reproduced without prior written consent.

Hive Variables (1)

- **Hive supports variable substitution**
  - Swaps a placeholder with a variable's literal value at run time
  - Variable names are case-sensitive
- **Within the Hive shell, set a named variable equal to some value:**

```
hive> SET state=CA;
```

- **To use the variable's value in a HiveQL query:**

```
hive> SELECT * FROM employees
      WHERE STATE = '${hiveconf:state}';
```


© Copyright 2010-2014 Cloudera. All rights reserved. Not to be reproduced without prior written consent.

Hive Variables (2)

- **You can set variables when you invoke Hive from the command line**
  - Eases repetitive queries by reducing need to modify HiveQL
- **For example, imagine that we have the following in `state.hql`**

```
SELECT COUNT(DISTINCT emp_id) FROM employees
        WHERE state = '${hiveconf:state}';
```

- **This makes creating per-state reports easy:**

```
$ hive -hiveconf state=CA -f state.hql > ca_count.txt
$ hive -hiveconf state=NY -f state.hql > ny_count.txt
$ hive -hiveconf state=TX -f state.hql > tx_count.txt
```


© Copyright 2010-2014 Cloudera. All rights reserved. Not to be reproduced without prior written consent.

Essential Points

- **SerDes govern how Hive reads and writes a table's records**
  - Specified (or defaulted) when creating a table
- **`TRANSFORM` processes records using an external program**
  - This can be written in nearly any language
- **UDFs are User-Defined Functions**
  - Custom logic that can be invoked just like built-in functions
- **Hive substitutes variable placeholders with literal values you assign**
  - This is done when you execute the query
  - Especially helpful with repetitive queries

© Copyright 2010-2014 Cloudera. All rights reserved. Not to be reproduced without prior written consent.